Contents

Overview

This document is intended for developers looking for more information regarding the LexEVS API.

Software Requirements

Information regarding the software requirements LexEVS can be found in the LexEVS 5.0 Installation Guide.

Setting up your Environment

Information regarding the installation and configuration of the LexEVS environment can be found in the LexEVS 5.0 Installation Guide.

LexEVS API

Programming interfaces for the system fall into three primary categories:

Core Services

Includes the LexBIGService, LexBIGServiceManager, CodedNodeSet and CodedNodeGraph classes, which provide the initial entry points for programmatic access to all system features and data.

Service Extensions

The extension mechanism provides for pluggable system features. Current extension points allow for the introduction of custom load and indexing mechanisms, unique query sort and filter mechanisms, and generic functional extensions which can be advertised for availability to client programs.

Utilities

Utility classes, such as those implementing iterator support, are provided by the system to provide convenience and optimize the handling of resources accessed through the runtime.

Core Services

Provides central entry points for programmatic access to system features and data.

LexBIGService

Components of interest include:

CodedNodeGraph

A virtual graph where the edges represent associations and the nodes represent concept codes. A

CodedNodeGraph describes a graph that can be combined with other graphs, queried or resolved into an actual graph rendering.

CodedNodeSet

A coded node set represents a flat list of coded entries.

LexBIGService

This interface represents the core interface to a LexEVS service.

LexBIGServiceManager

The service manager provides a single write and update access point for all of a service's content.

The service manager allows new coding schemes to be validated and loaded, existing coding schemes to be retired and removed and the status of various coding schemes to be updated and changed.

LexBIGServiceMetadata

Interface to perform system-wide query over optionally loaded metadata for loaded code systems and providers.

Service Extensions

Provides registration and lookup for pluggable system features.

Extensions

Components of interest include:

ExtensionRegistry

Allows registration and lookup of implementers for extensible pieces of the LexEVS architecture.

Extendable

Marks a class as an extension to the LexEVS application programming interface. This allows for centralized registration, lookup, and access to defined functions.

Query Extensions

Query extensions provide the ability to further constrain or manage query results.

Query

Components of interest include:

Filter

Allows for additional filtering of query results.

Sort

Allows for unique sorting of query results. This interface provides a comparator to evaluate order of any two given items from the result set.

Load Extensions

Load extensions are responsible for the validation and import of content to the LexEVS repository. Vocabularies may be imported from a variety of formats including LexGrid canonical XML, NCI Thesaurus (OWL), and NCI MetaThesaurus (UMLS RRF).

Load

Components of interest include:

Loader

The loader interface validates and/or loads content for a service.

LexGrid_Loader

Validates and/or loads content provided in the LexGrid canonical XML format.

NCI_MetaThesaurusLoader

Validates and/or loads the complete NCI MetaThesaurus. Content is supplied in RRF format. Note: To load individual coding schemes, consider using the UMLS_Loader as an alternative.

OBO_Loader

Validates and/or loads content provided in Open Biomedical Ontologies (OBO) text format.

OWL_Loader

Validates and/or loads content provided in Web Ontology Language (OWL) XML format. Note that for LexEVS phase 1 this loader is designed to specifically handle the NCI Thesaurus as provided in OWL format.

Text_Loader

A loader for delimited text type files. Text files come in one of two formats: indented code/designation pair or indented code/designation/description triples.

UMLS_Loader

Load one or more coding schemes from UMLS RRF format stored in a SQL database.

MetaData_Loader

Validates and/or loads content provided in metadata xml format. The only requirement of the xml file is that it be a valid xml file.

NCIHistoryLoader

A loader that takes the delimited NCI history file and applies it to a coding scheme.

OBOHistoryLoader

Load an OBO change history file.

Export Extensions

Export extensions are responsible for the export of content from the LexEVS repository to other representative vocabulary formats.

Export

Components of interest include:

Exporter

Defines a class of object used to export content from the underlying LexGrid repository to another repository or file format.

LexGrid_Exporter

Exports content to LexGrid canonical XML format.

OBO_Exporter

Exports content to OBO text format.

OWL_Exporter

Exports content to OWL XML format.

Index Extensions

Index extensions are built to optimize the finding, sorting and matching of query results.

Index

Components of interest include:

Index

Identifies expected behavior and an associated loader to build and maintain a named index. Note that a single loader may be used to maintain multiple named indexes.

IndexLoader

Manages registered index extensions. A single loader may be used to create and maintain multiple indexes over one or more coding schemes.

It is the responsibility of the loader to properly interpret each index it services by name, version, and provider.

Generic Extensions

Generic extensions provides a mechanism to register application-specific extensions for reference and reuse.

Generic

Components of interest include:

GenericExtension

The generic extension class. Classes that implement this class are accessible via the LexBIGService interface.

LexBIGServiceConvenienceMethods

Convenience methods to be implemented as a generic extension of the LexEVS API.

Utilities

Defines helper classes externalized by the LexEVS API.

Iterators

Iterators are used to provide controlled resolution of query results.

Iterators

Components of interest include:

EntityListIterator

Generic interface for flexible resolution of LexEVS objects.

ResolvedConceptReferencesIterator

An iterator for retrieving resolved coding scheme references.

Search Algorithms

Supported LexEVS Search Algorithms

Search Algorithm

```
Name: LuceneQuery
Version: 1.0
Description: Search with the Lucene query syntax.
See http://lucene.apache.org/java/2_3_2/queryparsersyntax.html)
```

Search Algorithm

```
Name: DoubleMetaphoneLuceneQuery
Version: 1.0
Description: Search with the Lucene query syntax, using a 'sounds like' algo
A search for 'atack' will get a hit on 'attack'
See http://lucene.apache.org/java/2_3_2/queryparsersyntax.html)
```

Search Algorithm

```
Name: StemmedLuceneQuery
Version: 1.0
Description: Search with the Lucene query syntax, using stemmed terms.
A search for 'trees' will get a hit on 'tree'
See http://lucene.apache.org/java/2_3_2/queryparsersyntax.html)
```

Search Algorithm

```
Name: startsWith
Version: 1.0
Description: Equivalent to 'term*' (case insensitive)
```

Search Algorithm

```
Name: exactMatch
Version: 1.0
Description: Exact match (case insensitive)
```

Search Algorithm

```
Name: contains
Version: 1.0
Description: Equivalent to '* term* *' - in other words - a trailing wildcar
(but no leading wild card) and the term can appear at any position.
```

Search Algorithm

```
Name: RegExp
Version: 1.0
Description: A Regular Expression query.  Searches against the lowercased te
regular expression that specifies an uppercase character will never return a
Additionally, this searches against the entire string as a single token, rat
the tokenized string - so write your regular expression accordingly.
Supported syntax is documented here:
http://jakarta.apache.org/regexp/apidocs/org/apache/regexp/RE.html
```

Additional Utility Classes

Note: It is highly recommended that all LexEVS programmers familiarize themselves with the classes contained in the `org.LexGrid.LexBIG.Utility` package. Many useful features are provided in an effort to increase approachability of the API and assist the programmer in common tasks. This package currently contains the following classes: ***Constructors*** ? Helper class to ease creating common objects. ***ConvenienceMethods*** ? One-stop shopping for convenience methods that have been implemented against the LexEVS API. ***LBConstants*** ? Provides constants for use in the LexEVS API. ***ObjectToString*** ? Provides centralized formatting of LexEVS Objects to String representations.

Code Examples

Concept Resolution

Programmers access coded concepts by acquiring first a node set or graph. After specifying optional restrictions, the nodes in this set or graph can be resolved as a list of `ConceptReference` objects which in turn contain references to one or more Concept objects. The following example provides a simple query of concept codes:

```
// Create a basic service object for data retrieval
LexBIGService lbSvc = new LexBIGServiceImpl();

// Create a concept reference list appropriate for this coding scheme an
// this concept code where the parameters are a String array consisting
// a single value and the name of the coding scheme where this concept r
ConceptReferenceList crefs = ConvenienceMethods.createConceptReferenceLi
                new String[] {code}, SAMPLE_SCHEME);

// Initialize a coding scheme version object with a version number for t
// sample scheme.
CodingSchemeVersionOrTag csvt = new CodingSchemeVersionOrTag();
csvt.setVersion(VERSION);

// Initialize a CodedNodeSet Object with all concepts in our sample codi
// scheme. (We named the scheme we wanted and by using the Boolean value
// false, retrieved both active and inactive concepts.) This method call
// ignores the version tag using the null parameter. The final
 // restrictToCodes(crefs) method call restricts the return to the singl
 // code in the previously initialized list of one.
CodedNodeSet nodes = lbSvc.getCodingSchemeConcepts(SAMPLE_SCHEME, csvt).
                restrictToCodes(crefs);

// Build a list of references from the current (and already restricted)
// and restrict them further to the single property of NCI_NAME and
// restrict to a single answer (parameter 1)).
ResolvedConceptReferenceList matches = nodes.resolveToList(
                null, ConvenienceMethods.createLocalNameList("FULL_SYN")

// Does our list of one contain the single reference we were looking for
// If so, then initialize a ResolvedConceptReference with the result and
// initialize a Concept object by calling the getReferencedEntry()
// method. The Concept object is the base information model object and
// contains, among other things, the CONCEPT_NAME value we were seeking.
// We retrieve it with a call to the first element in the properties lis
// getting the text && it's accompanying content.
if(matches.getResolvedConceptReferenceCount() <> 0)
{
```

```
        ResolvedConceptReference ref = (ResolvedConceptReference)matches.
                    enumerateResolvedConceptReference().nextElement();
        Concept entry = ref.getReferencedEntry();
        System.out.println("Matching synonym: " +
                entry.getPresentation(0).getValue() );
    }
    else
    {
        System.out.println("No match found");
    }
```

Service Metadata Retrieval

The LexEVS system maintains service metadata which can provide client programs with information about code system content and assigned copyright/licensing information. Below is an brief example showing how to access and print some of this metadata:

```
// We can get a CodingSchemeRenderingList object directly from LexBigServic
LexBIGService lbs = new LexBIGServiceImpl();
CodingSchemeRenderingList schemeList = lbs.getSupportedCodingSchemes();

for (CodingSchemeRendering csr : schemeList.getCodingSchemeRendering())
{
    CodingSchemeSummary css = csr.getCodingSchemeSummary();

    // Print separator then details from the CodingSchemeSummary
    System.out.println("=========================");
    System.out.println(ObjectToString.toString(css));

    // Set up a coding scheme reference to resolve Copyright
    String urn = css.getCodingSchemeURI();
    String version = css.getRepresentsVersion();
    CodingSchemeVersionOrTag csVorT =
              Constructors.createCodingSchemeVersionOrTagFromVersion(versi
    CodingScheme cs = lbs.resolveCodingScheme(urn, csVorT);
    System.out.println("Copyright: " +cs.getCopyright().getContent());

    // Get the final details from the RenderingDetail
    RenderingDetail rd = csr.getRenderingDetail();
    System.out.println(ObjectToString.toString(rd));
    System.out.println();
}
```

Combinatorial Queries

One of the most powerful features of the LexEVS architecture is the ability to define multiple search and sort criteria without intermediate retrieval of data from the LexEVS service. Consider the following code snippet:

```
System.out.println("Example double restriction query with additional "
        +"application of sort criteria and restricted return values.");
// Declare the service...
    LexBIGService lbs = new LexBIGServiceImpl();

    // Start with an unconstrained set of all codes for the vocabulary
CodingSchemeVersionOrTag csvt = new CodingSchemeVersionOrTag();
csvt.setVersion(VERSION);
CodedNodeSet cns = lbs.getCodingSchemeConcepts(SAMPLE_SCHEME, csvt);

// Constrain to concepts with designations (assigned text presentations
// that contain text that sounds like 'heart ventricle'
cns.restrictToMatchingDesignations(
                "hart ventrickle",
                SearchDesignationOption.ALL,
                MatchAlgorithms.DoubleMetaphoneLuceneQuery.toString(),
                null);

// Further restrict the results to concepts with a semantic type of
// 'Anatomical Structure'
cns.restrictToMatchingProperties(
                Constructors.createLocalNameList("Semantic_Type"),
                "Anatomical Structure",
                "exactMatch",
                null);

// Indicate that the resulting list should be sorted with the best
 // results first and then sorted by code if there is a tie.
SortOptionList sortCriteria = Constructors.createSortOptionList(
                new String[] {"matchToQuery", "code"});

// Indicate to return only the assigned UMLS_CUI and
 // textualPresentation properties.
LocalNameList restrictTo =ConvenienceMethods.createLocalNameList(
                new String[] {"UMLS_CUI", "textualPresentation"} );

// Still nothing computed yet.
// Perform the query && resolve the sorted/filtered list with a
 // maximum of 6 items returned.
ResolvedConceptReferenceList list = cns.resolveToList(
                sortCriteria, restrictTo, null, 6);
// Print the results
ResolvedConceptReference[] rcr = list.getResolvedConceptReference();
for (ResolvedConceptReference rc : rcr)
{
```

```
        System.out.println("Resolved Concept: +" +rc.getConceptCode());
    }
```

This example shows a simple yet powerful query to search a code system based on a ?sounds like? match algorithm (the list of all available match algorithms can be listed using the *?ListExtensions ?m?* admin script).

Declaring the target concept space

The coded node set (variable ?cns?) is initially declared to query the NCI Thesaurus vocabulary. At this point the concept space included by the set can be thought of as unrestricted, addressing every defined coded entry (the ?false? value on the declaration indicates to also include inactive concepts). However, it important to note that no search is performed by the LexEVS service at this time.

Applying filter criteria

Similarly, no computation is performed (to realize query results) during invocation of the `restrictToMatchingDesignations()` and `restrictToMatchingProperties()` methods. However, these calls effectively narrow the target space even further, indicating that filters should be applied to the information returned by the LexEVS query service.

Using the Lucene Query Syntax and other text matching functions

The text criteria applied in methods such as restrictToMatchingDesignations() uses one of a number of powerful text processing applications to provide the user with broad capability for text based searches. Text matches can be simple applications of exactMatch, startsWith or contains algorithms as well as powerful regular expressions and Lucene Query syntax (used in the LuceneQuery function.) As shown above these options are passed into the restrictToMatchingDesignations() Method as parameters.

Lucene Queries are well documented and can be very powerful. The uninitiated user may need some background on their use however. The user should start here with the official Lucene Query Parser documentation.

Keep in mind that some LexEVS queries such as "startsWith" and "contains" use wild card searches under the covers, so that use of wild cards in this context can cause errors in searches involving these search types.

Instead it is best to use the flexibility of the Lucene Query searches in the matchingDesignation by using the Lucene Query searches in LexEVS where most searches will work much as described in the query syntax documentation.

Special characters in the Lucene Query search can cause unexpected results. If you are not using special characters as recommended for various Lucene search mechanisms then your searches may not return expected results or may return an error. If the value you are searching upon contains say, parenthesis, you will need to place the value in quotations. The escape characters described in the Lucene Documentation do not work at this time.

Likewise you should not expect to see a Lucene Query narrow down search results as you progressively enter a longer substring more closely matching your term of interest. Instead use the contains method.

Applying sorting criteria

Multiple sort algorithms can be applied to control the order of items returned. In this case, we indicate that results are to be sorted based on primary and secondary criteria. The "matchToQuery" algorithm indicates to sort the result according to *best* match as determined by the search engine. The "code" item indicates to perform a secondary sort based on concept code.

Note: the list of all available sort algorithms can be listed using the *?ListExtensions ?s?* admin script.

Restricting the information returned for matching items

The LexEVS API also allows the programmer to restrict the values returned for each matching concept. In this example, we chose to return only the UMLS CUI and assigned text presentations.

Retrieving the result

A query is finally performed during the ?resolve? step, with results returned to the declared list. It is at this point that the LexEVS service does the heavy lifting. By declaring the full extent of the request up front (namespace, match criteria, sort criteria, and returned values), the service then has the opportunity to optimize the query path. In addition, in this example we restrict the number of items returned to a maximum of 6. This combined approach has the benefit of reducing server-side processing while minimizing the volume and frequency of traffic between the client program and the LexEVS service.

Note: While this section provides one example of combining criteria, this same pattern can be applied to many of the `CodedNodeSet` and `CodedNodeGraph` operations. It is strongly recommended that programmers familiarize themselves with this programming model and its application.

Additional Resources

The examples and automated test programs provided by the LexEVS installation (see file breakdown in Overview of the Software) are available as additional reference materials.

LexEVS GUI

The LexEVS Graphical User Interface, or GUI, is an optional component of the LexEVS install which will be in the /gui folder of the base LexEVS installation (see file breakdown in Overview of the Software). The GUI is meant to provide a simple tool to test LexEVS API methods and quickly view the results; almost all public methods defined by the LexEVS API are supported. This guide provides a brief overview of how the GUI can aid programmers in writing code to the LexEVS API.

Note: The LexEVS GUI supports both administrative and test functions. Please refer to the *LexEVS Administrator?s Guide* for instructions on using the GUI as an administration tool.

Launching the GUI

Depending on the operating systems that you selected at installation time, you should have one or more of the following programs in the `/gui` folder:

```
Linux_64-lbGUI.sh                Linux-lbGUI.sh
OSX-lbGUI.command                Windows-lbGUI.bat
```

Launch the GUI by executing the appropriate script for your platform. You will be presented with an application that looks like this:

Overview

The upper section of the GUI shows all of the code systems currently loaded, along with corresponding metadata. The lower section of the GUI is used to combine, restrict and resolve Code Sets and Code Graphs.

The lower left section is where you can perform Boolean logic on Code Sets and Code Graphs. The lower right section is where you can introduce restrictions on Code Sets and Code Graphs and browse results.

Note: The menu options are used primarily for administrative functions, and are covered in detail by the *LexEVS Administrator?s Guide*. In addition, all of the disabled buttons in the top half of the application are used for administrative functions, and are also described in the *LexEVS Administrator?s Guide*.

Creating New Queries

There are four buttons on the top half that are of interest for creating queries.

- **Refresh** ? This button causes the LexEVS GUI to reread the available terminologies and their respective metadata. This can be useful when using the GUI to view a LexEVS environment that is being modified by another process.
- **Get History** ? If a terminology with available history data is selected, this button opens a history browser to view it via the NCI history API. This option is currently only applicable when working with the NCI Thesaurus terminology.
- **Get Code Set** ?This button causes the selected terminology to be added to the lower left section of the GUI as a code set ? which is noted by a ?CS? prefix.
- **Get Code Graph** ?This button causes the selected terminology to be added to the lower left section of the GUI as a code graph ? which is noted by a ?CG? prefix.

Customizing Queries

After selecting a code system and clicking on **Get Code Set** or **Get Code Graph**, a row will be added to the lower left section of the GUI for each click. There are seven buttons in the lower left section that allow combinatorial logic between the code sets in the lower left.

- **Union** ? This button is enabled if two Code Sets or two Code Graphs are selected in the lower left. Clicking the button creates a new virtual Code Set or Code Graph which represents the Boolean union of the two selected items. All restrictions applied to the individual items still apply.
- **Intersection** ? This button is enabled if two Code Sets or two Code Graphs are selected in the lower left. Clicking the button creates a new virtual Code Set or Code Graph which represents the Boolean intersection of the two selected items. All restrictions applied to the individual items still apply.

- **Difference** ? This button is enabled if two Code Sets or two Code Graphs are selected in the lower left. Clicking the button creates a new virtual Code Set which represents the Boolean difference of the two selected Code Sets. All restrictions applied to the individual items still apply.
- **Restrict to Codes** ? This button is enabled if a Code Set and a Code Graph are selected in the lower left. Clicking the button creates a new virtual Code Graph which will be restricted to concept codes occurring in the selected Code Set.
- **Restrict to Source Codes** ? This button is enabled if a Code Set and a Code Graph are selected in the lower left. Clicking the button creates a new virtual Code Graph which will have its source codes restricted to codes occurring in the selected Code Set.
- **Restrict to Target Codes** ? This button is enabled if a Code Set and a Code Graph are selected in the lower left. Clicking the button creates a new virtual Code Graph which will have its target codes restricted to codes occurring in the selected Code Set.
- **Remove** ? This button is enabled if any Code Set or Code Graph (or virtual Code Set or Code Graph) is selected in the lower left. Clicking the button will remove the selected item from the list.

The lower right section of the GUI is used to apply restrictions to Code Sets or Code Graphs, and set the variables that need to be passed into the resolve method.

Working with Code Sets

If a Code Set is selected in the lower left, then the lower right section will look like this:

In the lower right section, there are two halves ? the top half and the bottom half. The top half is used to apply restrictions. The bottom half provides query options and resolution.

- **Add** ? This button introduces a new restriction to the Coded Node Set. Clicking it will bring up the following dialog box for creating restrictions:

The top drop down list indicates the type of restriction to add. The rest of the dialog box will change depending on the type of restriction selected. All required parameters for the selected restriction type will be presented.

- **Edit** ? This button is enabled when a restriction is selected. Clicking it allows revision of an existing restriction.
- **Remove** ?This button is enabled when a restriction is selected. Clicking it removes the selected restriction.
- **Only Include Active Codes** ? This check box indicates whether or not to include inactive codes when resolving the selected code set.
- **Set Sort Options** ? This button will bring up a dialog box to choose the desired sort order of the results.
- **Resolve Code Set** ? This button will bring up a result window where the Code Set will be resolved and displayed.

Working with Code Graphs

If you select a Coded Node Graph in the lower left section of the LexEVS GUI, the lower right section will look like this:

Again, there are two halves to the lower right section. The top half allows restrictions to be applied to the selected Code Graph, and it works the same as it does for a Coded Node Set. Please see the section above on applying restrictions to a Coded Node Set.

The lower half provides additional variables applicable when resolving a Coded Node Graph. For further explanation of these options, refer to the LexEVS API documentation.

- **Relation Container** (Optional) ? Indicates the CodingScheme Relations container to query. The drop down list is populated with allowable selections.
- **Focus Code** (Optional) ? Provides the code used as a starting point when resolving graph relations. This value is required for some queries, depending on the nature of requested associations.
- **Focus Code System** (Optional) ? Indicates the code system containing the Focus Code. The drop down list is populated with allowable selections.
- **Max Resolve Depth** ? How many levels deep should the graph be resolved? -1 is the default, which does not limit the depth.
- **Resolve Forward** ? Populate codes downstream from the focus node (based on directionality defined by each association).
- **Resolve Backward** ? Populate codes upstream from the focus node (based on directionality defined by each association).
- **Set Sort Options** ? This button will bring up a dialog box to choose the desired sort order of the results.
- **Resolve As Set** ? Resolves and displays the graph results as a coded node set.
- **Resolve As Graph** ?Resolves and displays the graph results.

Viewing Query Results

Clicking on the Resolve buttons for either a Coded Node Set or a Coded Node Graph will bring up the Result Browser window:

The left side shows a list of all the concept codes returned. When a concept code is selected on the left, the upper right will show a full description of the selected code. The lower right will show a graph view of the neighboring concepts.

When a Coded Node Graph is resolved, the result viewing window will look like this (this is the same Code System as above):

The left side still has a list of all of the concepts in the graph. The upper right will give a description of the selected concept. The lower right shows the entire graph.

The lower right section is adjustable, and dynamic. It responds to mouse clicks, dragging, and numerous key combinations. Beyond a depth of 3, the graph may ?collapse? and not show all of the nodes until you click on a node. Clicking on a node will cause it to expand out and display its children. Here are a list of key combinations recognized by the graph viewer:

- Left Click + Mouse Movement ? Drags the view.
- Right Click + Mouse Movement Up Or Down ? Zooms in or out.
- Right Click (on white space) ? Zooms the view to fit.
- Ctrl + ?+? ? Expands the graph connection lines
- Ctrl + ?-? ? Contracts the graph connection lines
- Ctrl + ?1? (or ?2? or ?3? or ?4?) ? Changes the orientation of the graph.

LexEVS caCORE Data Services API

Interacting with caCORE LexEVS

This chapter describes the components of the caCORE LexEVS and the service interface layer provided by the EVS API architecture. It gives examples of how to use the EVS APIs. It also describes the Distributed LexEVS API and the Distributed LexEVS APIAdapter.

caCORE LexEVS Components

The caCORE LexEVS API is a public domain, open source wrapper that provides full access to the LexEVS Terminology Server. LexEVS hosts the NCI Thesaurus, the NCI Metathesaurus, and several other vocabularies. Java clients accessing the NCI Thesaurus and Metathesaurus vocabularies communicate their requests via the open source caCORE LexEVS APIs, as shown in Overview of the caCORE LexEVS 4.0 release components.

Figure 4.1 - Overview of the caCORE LexEVS 4.0 release components

The open source interfaces provided as part of caCORE LexEVS 5.x include Java APIs, a SOAP interface, and an HTTP REST interface. The Java APIs are based on the EVS 3.2 object model and the LexEVS Service object model.

The EVS 3.2 model, exposed as part of caCORE 3.2, has been re-released with LexEVS as the back-end terminology service in place of the proprietary Apelon DTS back end. The SOAP and HTTP REST interfaces are also based on the 3.2 object model. The SDK 4.0 was used to generate the EVS 3.2 Java API, as well as the SOAP and HTTP REST interfaces.

The only difference between the EVS 3.2 API exposed as part of the caCORE LexEVS 5.x and the API exposed as part of caCORE 3.2 is the back-end terminology server used to retrieve the vocabulary data. The interface (API calls) are the same and should only require minor adjustments to user applications.

Note:	You cannot integrate caCORE 3.2 components with caCORE LexEVS 5.x. If you used multiple components of caCORE 3.2 (for example, EVS with caDSR), you need to continue to work with the caCORE 3.2 release until the other caCORE 4.0 components are available.

The LexEVS object model was developed by the Mayo Clinic. In its native form, the associated API assumes a local, non-distributed means of access. With caCORE LexEVS 5.x, a proxy layer enables EVS API clients to access the native LexEVS API from anywhere without having to worry about the underlying data sources. This is called the Distributed LexEVS (DLB) API.

The DLB Adapter is another option for caCORE LexEVS 5.x clients who choose to interface directly with the LexEVS API. This is essentially a set of convenience methods intended to simplify the use of the LexEVS API. For example, a series of method calls against the DLB API might equate to a single method call to the DLB Adapter.

Note:	The DLB Adapter is not intended to represent a complete set of convenience methods. As part of the caCORE LexEVS 5.x release, the intention is that users will work with the DLB API and suggest useful methods of convenience to the EVS Development Team.

LexEVS Data Sources

The LexEVS data source is the open source LexEVS terminology server. EVS clients interface with the LexEVS API to retrieve desired vocabulary data. The EVS provides the NCI with services and resources for controlled biomedical vocabularies, including the NCI Thesaurus and the NCI Metathesaurus.

NCI Thesaurus

The NCI Thesaurus is composed of over 27,000 concepts represented by about 78,000 terms. The Thesaurus is organized into 18 hierarchical trees covering areas such as Neoplasms, Drugs, Anatomy, Genes, Proteins, and Techniques. These terms are deployed by the NCI in its automated systems for uses such as key wording and database coding.

NCI Metathesaurus

The NCI Metathesaurus maps terms from one standard vocabulary to another, facilitating collaboration, data sharing, and data pooling for clinical trials and scientific databases. The Metathesaurus is based on the Unified Medical Language System (UMLS) developed by the National Library of Medicine (NLM). It is composed of over 70 biomedical vocabularies.

Interfaces

Main interfaces included in the LexEVSAPI package.

LexEVSDistributed

The Distributed LexEVS Portion of LexEVSAPI. This interface is a framework for calling LexEVS API methods remotely, along with enforcing security measures. JavaDoc

LexEVSDataService

The caCORE-SDK Data Service Portion of LexEVSAPI. This extends on the caCORE ApplicationService to provide additional Query Options. JavaDoc

LexEVSService

The Main LexEVSAPI Interface. This includes support for caCORE-SDK Data Service calls as well as remote LexBIG API calls. JavaDoc

Search Paradigm

The caCORE LexEVS architecture includes a service layer that provides a single, common access paradigm to clients that use any of the provided interfaces. As an object-oriented middleware layer designed for flexible data access, caCORE LexEVS relies heavily on strongly typed objects and an *object-in/object-out* mechanism.

Accessing and using a caCORE LexEVS system requires the following steps:

1. Ensure that the client application has access to the objects in the domain space.
2. Formulate the query criteria using the domain objects.
3. Establish a connection to the server.
4. Submit the query objects and specify the desired class of objects to be returned.

5. Use and manipulate the result set as desired.

caCORE LexEVS systems use four native application programming interfaces (APIs). Each interface uses the same paradigm to provide access to the caCORE LexEVS domain model, with minor changes specific to the syntax and structure of the clients. The following sections describe each API, identify installation and configuration requirements, and provide code examples.

The sequence diagram in Sequence diagram - caCORE 4.0 LexEVS API search mechanism illustrates the caCORE LexEVS API search mechanism implemented to access the NCI EVS vocabularies.

Figure 4.4 - Sequence diagram - caCORE 4.0 LexEVS API search mechanism

Querying the System

LexEVS conforms to the caCORE SDK API ? for more information see caCORE SDK 4.1 Programmer's Guide

QueryOptions

QueryOptions are designed to give the user extra control over the query before it is sent to the system. QueryOptions may be used to modify a query in these ways:

1. ?CodingScheme? ? Restricts the query to the specified Coding Scheme, instead of querying every available Coding Scheme.

2. CodingSchemeVersionOrTag? ? Restricts the query to the specified Version of the Coding Scheme. Note that:

    a. This may NOT be specified without also specifying the ?CodingScheme? attribute.
    b. If left unset, it will default to the version of the Coding Scheme tagged as ?PRODUCTION? in the system.

1. ?SecurityTokens? ? Security Tokens to use with the specified query. These Security Tokens are scoped to the current query ONLY. An subsequent queries will also need to specify the necessary Query Options.
2. ?LazyLoad? ? Some high use-case model Objects have bee ?lazy-load? enabled. This means that some attributes and associations of a model Object may not be fully populated when returned to the user. This allows for faster query times. This defaults to **false**, meaning that all attributes and associations will be eagerly fetched by the server and model Objects will always be fully populated. To enable this on applicable Objects, set to **true**.
   NOTE: Lazy Loading may only be used in conjunction with specifying a Coding Scheme and Version with the ?CodingScheme? and ?CodingSchemeVersionOrTag? attributes above.
3. ?ResultPageSize? ? the page size of results to return. The higher the number, the more results the system will return to the user at once. The client will request the next group of query results transparenly. This parameter is useful for performance tuning. For example, if a query returns a result of10,000 Objects, a ?ResultPageSize? of ?1000? would make 10 calls to the server returning a page of 1000 results each time. If left unset, this value will default to the default set Page Size

Examples of Use

Example 4.1: Query By Example with No Query Options

```
1  public static void main(String[] args)
2  {
3      try {
4          LexEVSApplicationService appService =
5            (LexEVSApplicationService)ApplicationServiceProvider.
6             getApplicationService("EvsServiceInfo");
7          Entity entity = new Entity()
8          entity.setEntityCode(?C1234?);
9          List<Entity> list = appService.search(Entity.class, entity);
10     } catch(ApplicationException ex){
11     }
12  }
```

Explanation of statements in explains specific statements in the code by line number .

Line Number	***Explanation***
4	Creates an instance of a class that implements the LexEVSApplicationService interface. This interface defines the service methods used to access data objects.

7	Construct the Query By Example Object and populate it with the desired search critieria. For this example, seach for any ?Entity? with an ?entityCode? attribute equaling ?C1234?.
9	Calls the search method of the LexEVSApplicationService object. This method returns a List Collection. This list will contain all of the ?Entity? Objects that match the search critieria. It this case, it will return all ?Entity? Objects with an ?entityCode? of ?C1234?.

Table 4.6 - Explanation of statements in :

Example 4.2 : Query By Example with Query Options

```
public static void main(String[] args)
{
   try {
      LexEVSApplicationService appService = (LexEVSApplicationService)Applicati
                                             getApplicationService("EvsServiceI
      QueryOptions queryOptions = new QueryOptions();
      queryOptions.setCodingScheme(?NCI Thesaurus?);
      CodingSchemeVersionOrTag csvt = new CodingSchemeVersionOrTag();
      csvt.setVersion(?09.10d?);
      queryOptions.setCodingSchemeVersionOrTag(csvt);
      Entity entity = new Entity();
      entity.setEntityCode(?C1234?);
      List&lt;Entity&gt; list = appService.search(Entity.class, entity, queryOp
   }
   catch(ApplicationException ex){
   }
}
```

Explanation of statements in explains specific statements in the code by line number.

Line Number	***Explanation***
4	Creates an instance of a class that implements the LexEVSApplicationService interface. This interface defines the service methods used to access data objects.
7	Construct the QueryOptions Object.
8	Populate the QueryOptions with the desired Coding Scheme.
9	Construct a CodingSchemeVersionOrTag Object.
10	Populate the CodingSchemeVersionOrTag Object with the desired Version.
11	Populate the QueryOptions with the above CodingSchemeVersionOrTag Object.
12	Construct the Query By Example Object and populate it with the desired search critieria. For this example, seach for any ?Entity? with an ?entityCode? attribute equaling ?C1234?.
14	Calls the search method of the LexEVSApplicationService object, along with the QueryOptions. This method returns a List Collection. This list will contain all of the ?Entity? Objects that match the search critieria, while being further modified by the QueryOptions. It this case, it will return all ?Entity? Objects with an ?entityCode? of ?C1234? belonging to the CodingScheme ?NCI

	Thesaurus? Version ?09.10d?.

Table 4.7 - Explanation of statements in :

Web Services API

The caCORE LexEVS Web Services API enables access to caCORE LexEVS data and vocabulary data from development environments where the Java API cannot be used, or where use of XML Web services is more desirable. This includes non-Java platforms and languages such as Perl, C/C++, .NET framework (C#, VB.Net), and Python.

The Web services interface can be used in any language-specific application that provides a mechanism for consuming XML Web services based on the Simple Object Access Protocol (SOAP). In those environments, connecting to caCORE LexEVS can be as simple as providing the end-point URL. Some platforms and languages require additional client-side code to handle the implementation of the SOAP envelope and the resolution of SOAP types.To view a list of packages that cater to different programming languages, visit http://www.w3.org/TR/SOAP/ and http://www.soapware.org/.

To maximize standards-based interoperability, the caCORE Web service conforms to the Web Services Interoperability Organization (WS-I) basic profile. The WS-I basic profile provides a set of non-proprietary specifications and implementation guidelines that enable interoperability between diverse systems. For more information about WS-I compliance, visit http://www.ws-i.org.

On the server side, Apache Axis is used to provide SOAP-based, inter-application communication. Axis provides the appropriate serialization and deserialization methods for the JavaBeans to achieve an application-independent interface. For more information about Axis, visit http://ws.apache.org/axis/.

Configuration

The caCORE/LexEVS WSDL file is located at http://lexevsapi.nci.nih.gov/lexevsapi50/services/lexevsapi50Service?wsdl. In addition to describing the protocols, ports, and operations exposed by the caCORE LexEVS Web service, this file can be used by a number of IDEs and tools to generate stubs for caCORE LexEVS objects. This enables code on different platforms to instantiate native objects for use as parameters and return values for the Web service methods. For more information on how to use the WSDL file to generate class stubs, consult the specific documentation for your platform.

The caCORE LexEVS Web services interface has a single end point called `lexevsapi50Service`, which is located at http://lexevsapi.nci.nih.gov/lexevsapi50/services/lexevsapi50Service. Client applications should use this URL to invoke Web service methods.

Building a Java SOAP Client

LexEVSAPI provides a tool to create a Java SOAP client capable of connecting to a LexEVSAPI SOAP service.

In the ./webServiceSoapClient contains a build.xml file that will construct a LexEVSAPI SOAP client. Before building, you may edit this build.xml file to customize the build process. Editable properties include ?wsdlURL? and ?webServiceNamespace?. An example configuration is below:

```
<property name="wsdlURL" value="http://bmidev4:8180/lexevsapi50/services/lexevs
<property name="webServiceNamespace" value="http://bmidev4:8180/lexevsapi50/ser
```

To build the client, use the command ?ant all? from the ./webServiceSoapClient directory.

XML-HTTP API

The caCORE LexEVS XML-HTTP API, based on the REST (Representational State Transfer) architectural style, provides a simple interface using the HTTP protocol. In addition to its ability to be invoked from most Internet browsers, developers can use this interface to build applications that do not require any programming overhead other than an HTTP client. This is particularly useful for developing Web applications using AJAX (Asynchronous JavaScript and XML).

Service Location and Syntax

The CORE EVS XML-HTTP interface uses the following URL syntax:

```
http://{server}/{servlet}?query={returnClass}&{criteria}
        &startIndex={index}
        &codingSchemeName={codingSchemeName}
        &codingSchemeVersion={codingSchemeVersion}
```

Table 4.12 explains the syntax, indicates whether specific elements are required, and gives examples.

Element	*Meaning*	*Required*	*Example*
`server`	Name of the Web server on which the caCORE LexEVS 5.0 Web application is	Yes	`lexevsapi.nci.nih.gov/lexevsapi50`

	deployed.		
servlet	URI and name of the servlet that will accept the HTTP GET requests.	Yes	lexevsapi50/GetXML lexevsapi50/GetHTML
returnClass	Class name indicating the type of objects that this query should return.	Yes	query=DescLogicConcept
criteria	Search request criteria describing the requested objects.	Yes	DescLogicConcept [@id=2]
index	Starting index of the result set.	No	startIndex=25
codingSchemeName	Restrict the query to a specific Coding Scheme Name.	No	codingSchemeName=NCI_Thesaurus
codingSchemeVersion	Restrict the query to a specific Coding Scheme Version.	No NOTE: Must be used in conjunction with a ?codingSchemeName?	codingSchemeVersion=09.12d

Table 4.12 - URL syntax used by the caCORE LexEVS XML-HTTP interface

The caCORE LexEVS architecture currently provides two servlets that accept incoming requests:

- *GetXML* returns results in an XML format that can be parsed and consumed by most programming languages and many document authoring and management tools.
- *GetHTML* presents result using a simple HTML interface that can be viewed by most modern Internet browsers.

Within the request string of the URL, the criteria element specifies the search criteria using XQuery-like syntax. Within this syntax, square brackets ([and]) represent attributes and associated roles of a class, the *at* symbol (@) signals an attribute name/value pair, and a forward slash character (/) specifies nested criteria.

Criteria statements in XML-HTTP queries generally use the following syntax (although you can also build more complex statements):

```
{ClassName}[@{attributeName}={value}] [@{attributeName}={value}]?

ClassName}[@{attributeName}={value}]/

{ClassName}[@{attributeName}={value}]/?
```

Table 4.13 explains the syntax for criteria statements and gives examples.

Parameter	*Meaning*	*Example*
`ClassName`	The name of a class.	`Entity`
`attributeName`	The name of an attribute of the return class or an associated class	`_entityCode`
`value`	The value of an attribute.	`C123*`

Table 4.13 - Criteria statements within XML-HTTP queries

Examples of Use

The examples in Table 4.14 demonstrate the usage of the XML-HTTP interface. In actual usage, these queries would either be submitted by a block of code or entered in the address bar of a Web browser.

Note that the servlet name *GetXML* in each of the examples can be replaced with *GetHTML* to view with layout and markup in a browser.

Query	http://evsapi.nci.nih.gov/evsapi41/GetXML?query=DescLogicConcept[_entityCode=C123*]
Semantic Meaning	Find all objects of type Entity that contain an ?entityCode? matching the pattern *?C123*?*.

Table 4.14 - XML-HTTP interface examples

Working with Result Sets

Because HTTP is a stateless protocol, the caCORE LexEVS server cannot detect the context of any incoming request. Consequently, each invocation of *GetXML* or *GetHTML* must contain all of the information necessary to retrieve the request, regardless of previous requests. Developers should consider this when working with the XML-HTTP interface.

Controlling the Start Index

To specify a specific start position in the result set, specify the *&startIndex* parameter. This will scroll to the desired position within the set of results.

Internal-Use Parameters

A number of parameters, such as &resultCounter, &pageSize, and &page, are used internally by the system and are not designed to be set by the user.

NOTE:

When specifying attribute values in the query string, note that use of the following characters generates an error: [ ] / \ # & %

Distributed LexEVS API

Overview

In place of the existing EVS 3.2 object model, caCORE LexEVS is making a gradual transition toward a pure LexEVS back-end terminology server and exposure of the LexEVS Service object model. caCORE 3.2 and earlier required a custom API layer between external users of the system and the proprietary Apelon Terminology Server APIs. With the transition to LexEVS, caCORE LexEVS can publicly expose the open source terminology service API without requiring a custom API layer.

Architecture

The LexEVS API is exposed by the LexEVS caCORE System for remote, distributed access (Figure 4.5). The caCORE System?s `LexEVSApplicationService` class implements the `LexBIGService` interface, effectively exposing LexEVS via caCORE.

Since in many cases the objects returned from the `LexBIGService` are not merely beans, but full-fledged data access objects (DAOs), the caCORE LexEVS client is configured to proxy method calls into the LexEVS objects and forward them to the caCORE server so that they execute within the LexEVS environment.

Figure 4.5 - DLB Architecture

The DLB environment will be configured on the caCORE LexEVS Server (http://lexevsapi.nci.nic.gov/lexevsapi50). This will give the server access to the LexEVS database and other resources. The client must therefore go through the caCORE LexEVS server to access any LexEVS data.

LexEVS Annotations

To address LexEVS DAOs, the LexEVS API integration incorporated the addition of (1) Java annotation marking methods that can be safely executed on the client side; and (2) classes that can be passed to the client without being wrapped by a proxy. The annotation is named `@lgClientSideSafe`. Every method in the LexEVS API that is accessible to the caCORE LexEVS user had to be considered and annotated if necessary.

Aspect Oriented Programming Proxies

LexEVS integration with caCORE LexEVS was accomplished using Spring Aspect Oriented Programming (AOP) to proxy the LexEVS classes and intercept calls to their methods. The caCORE LexEVS client wraps every object returned by the `LexBIGService` inside an AOP Proxy with advice from a `LexBIGMethodInterceptor` (?the interceptor?).

The interceptor is responsible for intercepting all client calls on the methods in each object. If a method is marked with the `@lgClientSideSafe` annotation, it proceeds normally. Otherwise, the object, method name, and parameters are sent to the caCORE LexEVS server for remote execution.

Figure 4.6 - Sequence diagram showing method interception

LexEVS API Documentation

The Mayo Clinic wrote the LexEVS 5.0 API. Documentation describing the LexEVS Service Model is available on the LexGRID Vocabulary Services for caBIG® GForge site at https://gforge.nci.nih.gov/frs/?group_id=14.

LexEVS Installation and Configuration

The DLB API is strictly a Java interface and requires Internet access for remote connectivity to the caCORE LexEVS server. Access to the DLB API requires access to the `lexevsapi-client.jar` file, available for download on the NCICB Web site. The `lexevsapi-client.jar` file needs to be available in the

classpath. For more information, see Installing and Configuring the LexEVS 5.0 Java API.

Example of Use

Example 4.6: Using the DLB API

The following code sample shows use of the DLB API to retrieve the list of available coding schemes in the LexEVS repository.

```
public class Test {
/** * Initialize program variables */

    private String codingScheme = null;
    private String version = null;

    LexBIGService lbSvc;

    public Test(String codingScheme, String version)  {
        //Set the LexEVS URL (for remote access)
        String evsUrl = ?http://lexevsapi.nci.nih.gov/lexevsapi50/http/remoteSe
        boolean isRemote = true;
        this.codingScheme = codingScheme;
        this.version = version;

        // Get the LexBIG service reference from LexEVS Application Service
        lbSvc =  (LexEVSApplicationService)ApplicationServiceProvider.getApplic

        // Set the vocabulary to work with
        Boolean retval = adapter.setVocabulary(codingScheme);

        codingSchemeMap = new HashMap();
        try {
            // Using the LexBIG service, get the supported coding schemes
            CodingSchemeRenderingList csrl = lbSvc.getSupportedCodingSchemes();

            // Get the coding scheme rendering
            CodingSchemeRendering[] csrs = csrl.getCodingSchemeRendering();

            // For each coding scheme rendering...
            for (int i=0; i<csrs.length; i++) {
                CodingSchemeRendering csr = csrs[i];

                // Determine whether the coding scheme rendering is active or n
                Boolean isActive = csr.getRenderingDetail().getVersionStatus().
                if (isActive != null && isActive.equals(Boolean.TRUE)) {
                    // Get the coding scheme summary
                    CodingSchemeSummary css = csr.getCodingSchemeSummary();
```

```
                    // Get the coding scheme formal name
                    String formalname = css.getFormalName();

                    //Get the coding scheme version
                    String representsVersion = css.getRepresentsVersion();
                    CodingSchemeVersionOrTag vt = new;
                    CodingSchemeVersionOrTag();
                    vt.setVersion(representsVersion);

                    // Resolve coding scheme based on the formal name
                    CodingScheme scheme = null;

                    try {
                        scheme =lbSvc.resolveCodingScheme(formalname, vt);
                        if (scheme != null){
                            codingSchemeMap.put((Object) formalname, (Object) s
                        }
                    } catch (Exception e) {
                        // Resolve coding scheme based on the URI
                        String uri = css.getCodingSchemeURI();
                        try {
                            scheme = lbSvc.resolveCodingScheme(uri, vt);
                            if (scheme != null) {
                                 codingSchemeMap.put((Object) formalname, (Obje
                            }
                        } catch (Exception ex) {
                            String localname = css.getLocalName();

                            // Resolve coding scheme based on the local name
                            try {
                                scheme = lbSvc.resolveCodingScheme(localname, v
                                if(scheme != null){
                                    codingSchemeMap.put((Object) formalname, (O
                                }
                            } catch (Exception e2) {
                            }
                        }
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /** *Main */
    public static void main (String[] args)
    {
        String name = ?NCI Thesaurus?;
        String version = ?06.12d?;
```

```
        // Instantiate the Test Class
        Test test = new Test(name, version);
    }
}
```

LexEVS Analytical Grid Service API

The following table summarizes the operations available through the LexEVS Analytical Grid Service. Each of the operations is also defined in detail below. The grid analytical service and related operations are viewable via the caGrid Portal (http://cagrid-portal.nci.nih.gov).

Using the API

There are two (2) different interfaces for accessing the LexEVS Grid Services:

1. org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter, or
2. org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter

Option 1, org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter provides an interface for interacting with the LexEVS Grid Services. This Interface is intended to mirror the existing LexEVS API as much as possible. There is no object wrapping for semantic purposes on this interface. This allows existing applications of the LexEVS API to use Grid Services without code changes.

This Interface may be acquired by instantiating LexBIGServiceAdapter with the Grid Service URL as a parameter.

```
LexBIGService lbs = new LexBIGServiceAdapter("http://lexevsapi-analytical50.nci
```

Option 2, org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter also provides an interface for interacting with the LexEVS Grid Services. However, this Interfaces is the semantically defined interface. All method parameters and return values are defined and annotated as CDEs to be loaded into caDSR. This Interface is intended to be caGrid Silver Level Compliant.

This Interface may be acquired by instantiating LexBIGServiceGridAdapter with the Grid Service URL as a parameter.

```
LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter("http://lexevsapi-analytic
```

Method Descriptions

getCodingSchemeConcepts

Description:	Returns the set of all (or all active) concepts in the specified coding scheme.
Input:	*org.LexGrid.LexBIG.DataModel.cagrid.CodingSchemeIdentification, org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag*
Output:	*org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.CodedNodeSet.stubs.types.CodedNodeSetRefere*
Exception:	*RemoteException*

Implementation Details:	***Implementation:***
	
	Step 1: Create a Resource on the server and populate it with the requested org.LexGrid.LexBIG.LexBIGService.CodedNodeSet.
	
	Step 2: Return the Client Reference to the user. This Reference has the above org.LexGrid.LexBIG.LexBIGService.CodedNodeSet as a Resource. An org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.service.CodedNodeSetClient object is built fron the above Reference.
	
	Sample Call:
	
	Step 1: Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter or org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter
	
	LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url);
	
	Step 2: Build a org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag containing the Version information for the desired Coding Scheme
	
	CodingSchemeVersionOrTag csvt = **new** CodingSchemeVersionOrTag();
	csvt.setVersion("testVersion");
	
	Step 3: Build an org.LexGrid.LexBIG.DataModel.cagrid.CodingSchemeIdentification to hold the Coding Scheme name.
	
	CodingSchemeIdentification codingScheme = **new** CodingSchemeIdentification();
	codingScheme.setCode(code);
	
	Step 4: Invoke the LexBIG caGrid service as follows: CodedNodeSetGrid cns = lbs.getCodingSchemeConcepts(codingScheme, csvt);

getFilter

Description:	Returns an instance of the filter extension registered with the given name.
Input:	*org.LexGrid.LexBIG.DataModel.cagrid.ExtensionIdentification*
Output:	*org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.Filter.stubs.types.FilterReference*
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Create a Resource on the server and populate it with the requested org.LexGrid.LexBIG.Extensions.Query.Filter *Step 2:* Return the Client Reference to the user. This Reference has the above org.LexGrid.LexBIG.Extensions.Query.Filter as a Resource. This client is a Service Context that allows the user to call regular org.LexGrid.LexBIG.Extensions.Query.Filter API calls through the grid service. An org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.Filter.client.FilterClient object is built from the above Reference. This FilterClient implements the Interface org.LexGrid.LexBIG.Extensions.Query.Filter. This makes calling Grid Service Calls through org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.Filter.client.FilterClient transparent to the end user. ***Sample Call:*** Step 1: Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter or org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter LexBIGServiceGrid lbs = **new** LexBIGServiceGridAdapter(url); Step 2: Build an org.LexGrid.LexBIG.DataModel.cagrid.ExtensionIdentification to hold the Extension name. ExtensionIdentification extension = **new** ExtensionIdentification(); extension.setLexBIGExtensionName(name); *Step 3:* Invoke the LexEVS caGrid service as follows: Filter filter = lbs.getFilter(extension);

getSortAlgorithm

Description:	Returns an instance of the sort extension registered with the given name.
Input:	*org.LexGrid.LexBIG.DataModel.cagrid.ExtensionIdentification*
Output:	*org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.Sort.* stubs.types.SortReference
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Create a Resource on the server and populate it with the requested org.LexGrid.LexBIG.Extensions.Query.Sort *Step 2:* Return the Client Reference to the user. This Reference has the above org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.Sort.client.SortClient as a Resource. This client is a Service Context that allows the user to call regular org.LexGrid.LexBIG.Extensions.Query.Sort API calls through the grid service. An org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.Sort.client.SortClient object is built from the above Reference. This SortClient implements the Interface org.LexGrid.LexBIG.Extensions.Query.Sort. This makes calling Grid Service Calls through org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.Sort.client.SortClient transparent to the end user. ***Sample Call:*** *Step 1:* Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter or org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url); *Step 2:* Build an org.LexGrid.LexBIG.DataModel.cagrid.ExtensionIdentification to hold the Extension name. ExtensionIdentification extension = **new** ExtensionIdentification(); extension.setLexBIGExtensionName(name); *Step 3:* Invoke the LexEVS caGrid service as follows: Filter filter = lbs.getSortAlgorithm(extension);

getFilterExtensions

Description:	Returns a description of all registered extensions used to provide additional filtering of query results.
Input:	*none*
Output	*org.LexGrid.LexBIG.DataModel.Collections.ExtensionDescriptionList*
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Call this method on the associated LexEVS Service instance (or Distributed LexEVS instance) on the server, and forward the results. ***Sample Call:*** *Step 1:* Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter or org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url); *Step 2:* Invoke the LexEVS caGrid service as follows: ExtensionDescriptionList extDescList = lbs.getFilterExtensions();

getServiceMetadata

Description:	Return an interface to perform system-wide query over metadata for loaded code systems and providers.
Input:	*none*
Output:	*org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.LexBIGServiceMetadata.* *stubs.types.LexBIGServiceMetadataReference*
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Create a Resource on the server and populate it with the requested org.LexGrid.LexBIG.LexBIGService.LexBIGServiceMetadata *Step 2:* Return the LexBIGServiceMetadataClient to the user. This LexBIGServiceMetadataClient has the above org.LexGrid.LexBIG.LexBIGService.LexBIGServiceMetadata as a Resource. An org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.service.LexBIGServiceMetadataClient object is built from the above Reference.

	Sample Call:' *Step 1:* Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter or org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url); *Step 2:* Invoke the LexEVS caGrid service as follows: LexBIGServiceMetadataGrid metadata = lbs.getServiceMetadata();

getSupportedCodingSchemes

Description:	Return a list of coding schemes and versions that are supported by this service, along with their status.
Input:	*none*
Output:	*org.LexGrid.LexBIG.DataModel.Collections.CodingSchemeRenderingList*
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Call this method on the associated LexEVS Service instance (or Distributed LexEVS instance) on the server, and forward the results. ***Sample Call:*** *Step 1:* Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter or org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url); *Step 2:* Invoke the LexEVS caGrid service as follows: CodingSchemeRenderingList csrl = lbs.getSupportedCodingSchemes();

getLastUpdateTime

Description:	Return the last time that the content of this service was changed; null if no changes have occurred. Tag assignments do not count as service changes for this purpose.

Input:	*none*
Output:	*java.util.Date*
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Call this method on the associated LexEVS Service instance (or Distributed LexEVS instance) on the server, and forward the results. ***Sample Call:*** *Step 1:* Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter or org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url); *Step 2:* Invoke the LexEVS caGrid service as follows: Date date = lbs.getLastUpdateTime();

resolveCodingScheme

Description:	Return detailed coding scheme information given a specific tag or version identifier.
Input:	*org.LexGrid.LexBIG.DataModel.cagrid.CodingSchemeIdentification, org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag*
Output:	*org.LexGrid.codingSchemes.CodingScheme*
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Call this method on the associated LexEVS Service instance (or Distributed LexEVS instance) on the server, and forward the results. ***Sample Call:*** *Step 1:* Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter or org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url); *Step 2:* Build an org.LexGrid.LexBIG.DataModel.cagrid.CodingSchemeIdentification to hold the Coding Scheme name. CodingSchemeIdentification codingScheme = **new** CodingSchemeIdentification();

	codingScheme.setCode(code);
	
	Step 3: Build a org.LexGrid.LexBIG.DataModel.Core
	containing the Version information for the desired Co
	
	CodingSchemeVersionOrTag csvt = new CodingSche
	csvt.setVersion("testVersion");
	
	Step 4: Invoke the LexEVS caGrid service as follows:
	lbs.resolveCodingScheme(codingScheme, csvt);

getNodeGraph

Description:	Returns the node graph as represented in the particular
Input:	*org.LexGrid.LexBIG.DataModel.cagrid.CodingScheme* *org.LexGrid.LexBIG.DataModel.Core.CodingSchemeV* *org.LexGrid.LexBIG.DataModel.cagrid.RelationConta*
Output:	*org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.* CodedNodeGraph.stubs.types.CodedNodeGraphRefere
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Create a Resource on the server and populate it org.LexGrid.LexBIG.LexBIGService.CodedNodeGraph *Step 2:* Return the Client Reference to the user. This Re org.LexGrid.LexBIG.LexBIGService.CodedNodeGraph org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.se is built from the above Reference. ***Sample Call:*** *Step 1:* Connect to the LexBIG caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceA org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceG LexBIGServiceGrid lbs = new LexBIGServiceGridAda *Step 2:* Build an org.LexGrid.LexBIG.DataModel.cagri hold the Coding Scheme name. CodingSchemeIdentification codingScheme = **new** Cod codingScheme.setCode(code);

	Step 3: Build an org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag containing the Version information for the desired Coding Scheme
	
	CodingSchemeVersionOrTag csvt = **new** CodingSchemeVersionOrTag();
	csvt.setVersion("testVersion");
	
	Step 4: Build an org.LexGrid.LexBIG.DataModel.cagrid.RelationContainerIdentification containing the Relation Container information.
	
	RelationContainerIdentification container = **new** RelationContainerIdentification();
	container.setDc(name);
	
	Step 5: Invoke the LexEVS caGrid service as follows, providing String parameters for the desired Coding Scheme and Relationship Name: CodedNodeGraphGrid cng = client.getNodeGraph(codingScheme, csvt, container);

getMatchAlgorithms

Description:	Returns the node graph as represented in the particular relationship set in the coding scheme.
Input:	*none*
Output:	*org.LexGrid.LexBIG.DataModel.Collections.ModuleDescriptionList*
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Call this method on the associated LexEVS Service instance (or Distributed LexEVS instance) on the server, and forward the results. ***Sample Call:*** *Step 1:* Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter or org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url); *Step 2:* Invoke the LexEVS caGrid service as follows: ModuleDescriptionList mdl = lbs.getMatchAlgorithms();

getGenericExtensions

Description:	Returns a description of all registered extensions used to implement application-specific behavior that is centrally accessible from a LexBIGService. Note that only generic extensions (base class GenericExtension) will be listed here. All other classes are retrievable at the appropriate interface point (filter, sort, etc).
Input:	*none*
Output:	*org.LexGrid.LexBIG.DataModel.Collections.ExtensionDescriptionList*
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Call this method on the associated LexEVS Service instance (or Distributed LexEVS instance) on the server, and forward the results. ***Sample Call:*** *Step 1:* Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter or org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url); *Step 2:* Invoke the LexEVS caGrid service as follows: ExtensionDescriptionList edl = lbs.getGenericExtensions();

getGenericExtension

Description:	Returns an instance of the application-specific extension registered with the given name.
Input:	*org.LexGrid.LexBIG.DataModel.cagrid.ExtensionIdentification*
Output:	*org.LexGrid.LexBIG.DataModel.Collections.SortDescriptionList*
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Call this method on the associated LexEVS Service instance (or Distributed LexEVS instance) on the server, and forward the results. ***Sample Call:*** *Step 1:* Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter or org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter

	LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url); NOTE: Currently this method will return a LexBIGServiceConvenienceMethods instance. *Step 2:* Build an org.LexGrid.LexBIG.DataModel.cagrid.ExtensionIdentification to hold the Extension name. ExtensionIdentification extension = **new** ExtensionIdentification(); extension.setLexBIGExtensionName(?LexBIGServiceConvenienceMethods?); *Step 3: Invoke the LexEVS caGrid service as follows:* LexBIGServiceConvenienceMethodsGrid lbscm = lbs.getGenericExtensions(extension); *Step 4:* Return the LexBIGServiceConvenienceMethodsClient to the user. This LexBIGServiceConvenienceMethodsClient has the above org.LexGrid.LexBIG.Extensions.Generic.LexBIGServiceConvenienceMethods as a Resource. An org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.service.CodedNodeGraphClient object is built from the above Reference.

getHistoryService

Description:	Resolve a reference to the history api servicing the given coding scheme.
Input:	*org.LexGrid.LexBIG.DataModel.cagrid.CodingSchemeIdentification*
Output:	*org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.* *HistoryService.stubs.types.HistoryServiceReference*
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Call this method on the associated LexEVS Service instance (or Distributed LexEVS instance) on the server, and forward the results. *Step 2:* Return the HistoryServiceClient to the user. This HistoryServiceClient has the above org.LexGrid.LexBIG.History.HistoryService as a Resource. This Client is a Service Context that allows the user to call regular org.LexGrid.LexBIG.History.HistoryService API calls through the grid service. HistoryServiceClient implements the Interface org.LexGrid.LexBIG.History.HistoryService. This makes calling Grid Service Calls through org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.HistoryService.client.HistoryServiceClient transparent to the end user. ***Sample Call:*** *Step 1:* Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter or org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter

	LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url); *Step 2:* Build an org.LexGrid.LexBIG.DataModel.cagrid.CodingSchemeIdentification to hold the Coding Scheme name. CodingSchemeIdentification codingScheme = **new** CodingSchemeIdentification(); codingScheme.setCode(code); *Step 3:* Invoke the LexEVS caGrid service as follows: HistoryServiceGrid history = lbs.getHistoryService(codingScheme);

getSortAlgorithms

Description:	Returns a description of all registered extensions used to provide additional filtering of query results.
Input:	*org.LexGrid.LexBIG.DataModel.InterfaceElements.types.SortContext*
Output:	*org.LexGrid.LexBIG.DataModel.Collections.SortDescriptionList*
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Call this method on the associated LexEVS Service instance (or Distributed LexEVS instance) on the server, and forward the results. ***Sample Call:*** Step 1: Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceAdapter or org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url); Step 2: Invoke the LexEVS caGrid service as follows: SortDescriptionList sortDescList = lbs.getSortAlgorithms(sortContext);

resolveCodingSchemeCopyright

Description:	Return coding scheme copyright given a specific tag or version identifier.
Input:	*org.LexGrid.LexBIG.DataModel.cagrid.CodingSchemeIdentification*
Output:	*org.LexGrid.LexBIG.DataModel.cagrid.CodingSchemeCopyRight*
Exception:	*RemoteException*

Implementation Details:	***Implementation:***
	
	Step 1: Call this method on the associated LexEVS Se
	LexEVS instance) on the server, and forward the resul
	
	Sample Call:
	
	Step 1: Connect to the LexEVS caGrid Service using t
	org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceA
	org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceC
	
	LexBIGServiceGrid lbs = new LexBIGServiceGridAd
	
	Step 2: Build an org.LexGrid.LexBIG.DataModel.cag
	hold the Coding Scheme name.
	
	CodingSchemeIdentification codingScheme = **new**
	
	`CodingSchemeIdentification();`
	
	codingScheme.setCode(code);
	
	Step 3: Build an org.LexGrid.LexBIG.DataModel.Cor
	containing the Version information for the desired Co
	
	CodingSchemeVersionOrTag csvt = **new** CodingSche
	csvt.setVersion("testVersion");
	
	Step 4: Invoke the LexEVS caGrid service as follows:
	= lbs.resolveCodingSchemeCopyright(codingScheme,

setSecurityToken

Description:	Sets the Security Token for the given Coding Scheme.
Input:	*org.LexGrid.LexBIG.DataModel.cagrid.CodingScheme*
Output:	*org.LexGrid.LexBIG.cagrid.LexEVSGridService.stubs.t*
Exception:	*RemoteException*
Implementation Details:	***Implementation:*** *Step 1:* Call this method on the associated LexEVS Serv forward the results. ***Sample Call:***

	Step 1: Connect to the LexEVS caGrid Service using the org.LexGrid.LexBIG.cagrid.adapters.LexBIG org.LexGrid.LexBIG.cagrid.adapters.LexBIGServiceGridAdapter LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url); *Step 2:* Build an org.LexGrid.LexBIG.DataModel.cagrid.CodingSchemeIdentification to hold the Cod CodingSchemeIdentification codingScheme = **new** CodingSchemeIdentification(); codingScheme.setName(?codingScheme?); *Step 3:* Build an gov.nih.nci.evs.security.SecurityToken containing the security information for the de Scheme. SecurityToken metaToken = **new** SecurityToken(); metaToken.setAccessToken(?token?); *Step 4:* Invoke the LexEVS caGrid service as follows: This will return a reference to a new ?LexBIGS instance that is associated with the security properties that were passed in. LexBIGServiceGrid lbsg = lbs.setSecurityToken(codingScheme, metaToken);

Usage Instructions

Service URL

The LexEVS Grid Service 4.2 URL is: http://lexevsapi.nci.nih.gov/wsrf/services/cagrid/LexEVSGridService.

The service is also accessible via the caGRID Portal.

Required Libraries

The libraries required for programmatic access to the LexEVS Grid Service are listed in the tables below. The 3rd Party Software Libraries required for use of the LexEVS API Grid Service are listed in Table 4.1 and the NCICB software captured under the caBIG® umbrella are listed in Table 4.2.

Table 4.1 3rd Party Libraries

Product	Jars	License	Hon

Apache WS-Addressing	addressing-1.0.jar	adressing 1.0.LICENSE	From Globus 4.0.2 Jav directory: http://www.globus.org Source available at http://ws.apache.org/a
Apache Axis	axis-ant.jar axis.jar commons-pool-1.3.jar commons-logging-1.1.jar commons-lang-2.2.jar commons-collections-3.2.jar commons-codec-1.3.jar log4j-1.2.8.jar jaxrpc.jar saaj.jar wsdl4j.jar	axis-jars.LICENSE	http://ws.apache.org/a
Apache Xerces	xercesImpl.jar	xerces.LICENSE	http://xerces.apache.or
Apache Lucene	lucene-core-2.3.2.jar lucene-regex-2.3.2.jar lucene-snowball-2.3.2.jar	Lucene LICENSE	http://lucene.apache.or
ASM - all purpose Java bytecode manipulation and analysis framework	asm.jar	http://asm.objectweb.org/license.html	http://asm.objectweb.c
Castor	castor-1.2.jar	http://www.castor.org/license.html	http://www.castor.org/
Globus Toolkit	cog-axis.jar cog-jglobus.jar	http://www.globus.org/toolkit/legal/4.0/	
Bouncy Castle Crypto APIs	jce-jdk13-125.jar	http://www.bouncycastle.org/licence.html	http://www.bouncycas
Open Permis	wsrf_core.jar wsrf_core_stubs.jar	http://www.openpermis.org/BSDlicenceKent.txt	http://www.openpermi
Apache	wss4j.jar	http://ws.apache.org/wss4j/license.html	http://ws.apache.org/w

WSS4J			
Spring	spring.jar	Spring LICENSE	http://www.springfram

Table 4.2 NCICB/caBIG Libraries

Library	Associated JARs
caGrid Software Libraries	caGrid-ServiceSecurityProvider-client-1.2.jar
	caGrid-ServiceSecurityProvider-common-1.2.jar
	caGrid-ServiceSecurityProvider-stubs-1.2.jar
	caGrid-core-1.2.jar
	caGrid-metadata-common-1.2.jar
	caGrid-metadata-data-1.2.jar
	caGrid-metadata-security-1.2.jar
	caGrid-metadatautils-1.2.jar
EVS API Libaries	evsapi42-beans.jar
	evsapi42-framework.jar
LexEVS Grid Service Client Library	LexEVSGridService-client.jar
LexEVS Grid Service Stubs	LexEVSGridService-stubs.jar
LexEVS Grid Service Common	LexEVSGridService-common.jar
LexEVS Grid Service Service	LexEVSGridService-service.jar
LexEVS Grid Service Tests	LexEVSGridService-tests.jar
caCORE SDK Library	sdk-client-framework.jar
LexEVS API	lexbig.jar
Custom Castor Serializer	castor-bean-serializer.jar

Downloads

For your convenience, the required libraries are available for download here:

https://gforge.nci.nih.gov/docman/view.php/491/14401/lexevs42-gridsrvc-jars.jar.

In order to programmatically access the LexEVS API Grid Service, these libraries need to be added to your local classpath.

Code Examples

Example client and service calls, and SOAP messages

See http://gforge.nci.nih.gov/docman/view.php/491/14252/TestClient.zip

Example API usage

Example 1: Searching for concepts in NCI Thesaurus containing the string ?Gene?

```
//Create a Connection to the Grid Service
LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(gridServiceURL);

//Set up the CodingSchemeIdentification object to define the Coding Scheme<
CodingSchemeIdentification csid = new CodingSchemeIdentification();
csid.setName("NCI Thesaurus");

//Get the CodedNodeSet for that CodingScheme (This returns a CodedNodeSet S
CodedNodeSetGrid cnsg = lbs.getCodingSchemeConcepts(csid, null);
//getCodingSchemeConcepts is a Grid Service Call

//Set the text to match
MatchCriteria matchText = new MatchCriteria();
matchText.setText("Gene");
//Define a SearchDesignationOption, if any
SearchDesignationOption searchOption = new SearchDesignationOption();

//Choose an algorithm to do the matching
ExtensionIdentification matchAlgorithm = new ExtensionIdentification();
matchAlgorithm.setLexBIGExtensionName("contains");

//Chose a language
LanguageIdentification language = new LanguageIdentification();
language.setIdentifier("en");

//Restrict the CodedNodeSet
cnsg.restrictToMatchingDesignations(matchText, searchOption, matchAlgorithm
//restrictToMatchingDesignations is a Grid Service Call

//Create a SetResolutionPolicy to handle the details of Resolving the Coded
//Here, we will set the Maximum number of Concepts returned to 10.
SetResolutionPolicy resolvePolicy = new SetResolutionPolicy();
resolvePolicy.setMaximumToReturn(10);

//Do the resolve
```

```
    ResolvedConceptReferenceList rcrlist = cnsg.resolveToList(resolvePolicy);
    //resolveToList is a Grid Service Call

    //Use the returned ResolvedConceptReferenceList to print some details about
    ResolvedConceptReference[] rcref = rcrlist.getResolvedConceptReference();
    for (int i = 0; i < rcref.length; i++) {
System.out.println(rcref[i].getConceptCode());
System.out.println(rcref[i].getReferencedEntry().
getPresentation()[0].getText().getContent());

    }
```

Error Handling

Error Connecting to LexEVS Grid Service

When connecting through the Java Client, java.net.ConnectException and org.apache.axis.types.URI.MalformedURIException may be thrown upon an unsuccessful attempt to connect.

A MalformedURIException is thrown in the case if a poorly-formed URL string. In this case, the exception is thrown before an attempt to connect is even made.

If the URL is well-formed, proper connection is tested. If the connection attempt fails, a ConnectException is thrown containing the reason for the failure.

```
    try{
        LexBIGServiceGridAdapter lbsg = new LexBIGServiceGridAdapter
            ("http://localhost:8080/wsrf/services/cagrid/LexEVSGridService");
    } catch(java.net.ConnectException e){
        //Error Connecting
        e.printStackTrace();
    } catch(org.apache.axis.types.URI.MalformedURIException e){
        //URL Syntax Error
        e.printStackTrace();
    }
```

This example shows a typical connection to the LexEVS Grid Service, with the two potential Exceptions being caught and handled as necessary.

LexEVS Errors

LexEVS errors will be forwarded through the Distributed LexEVS layer and then on to the Grid layer. Input parameters, along with any other LexEVS (or Distributed LexEVS) errors will be detected on the server, not the client, and forwarded. All Generic LexEVS (or Distributed LexEVS) errors will be forwarded via a RemoteException, with the cause of the error and underlying LexEVS error message included.

Invalid Service Context Access

Service Context Services are not meant to be called directly. If the client attempts to do so, an org.LexGrid.LexBIG.cagrid.LexEVSGridService.CodedNodeSet.stubs.types.InvalidServiceContextAccess Exception will be thrown. This indicates a call was made to a Service Context without obtaining a Service Context Reference via the Main Service (see the above section Service Contexts and State for more information).

Security Issues

LexEVS Grid Service Security

Certain vocabulary content accessible through the LexEVS Grid Service may require extra authorization to access. Each client is required to supply its own access credentials via Security Tokens. These Security Tokens are implemented by a SecurityToken object:

Name: SecurityToken Namespace: gme://caCORE.caCORE/3.2/gov.nih.nci.evs.security Package: gov.nih.nci.evs.security

Accessing Secure Content

A client establishes access to a secured vocabulary via the following Grid Service Calls:

Step 1: Connect to the LexEVS caGrid Service LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url);

Step 2: Build an org.LexGrid.LexBIG.DataModel.cagrid.CodingSchemeIdentification to hold the Coding Scheme name.

```
CodingSchemeIdentification codingScheme = new CodingSchemeIdentification();
codingScheme.setName(?codingScheme?);
```

Step 3: Build an gov.nih.nci.evs.security.SecurityToken containing the security information for the desired Coding Scheme.

SecurityToken token = **new** SecurityToken (); token.setAccessToken(?securityToken?);

Step 4: Invoke the LexEVS caGrid service as follows: This will return a reference to a new ?LexBIGServiceGrid? instance that is associated with the security properties that were passed in.

LexBIGServiceGrid lbsg = lbs.setSecurityToken(codingScheme, token);

It is important to note that the Grid Service ?setSecurityToken? returns an *org.LexGrid.LexBIG.cagrid.LexEVSGridService.stubs.types* .LexEVSGridServiceReference.LexEVSGridServiceReference object. This reference must be used to access the secured vocabularies.

Implementation

Each call to ?setSecurityToken? sets up a secured connection to Distributed LexEVS with the access privileges included in the SecurityToken parameter. The LexEVSGridServiceReference that is returned to the client contains a unique key identifier to the secure connection that has been created on the server. All subsequent calls the client makes through this LexEVSGridServiceReference will be made securely. If additional SecurityTokens are passed in through the ?setSecurityToken? Grid Service, the additional security will be added and maintained.

The ?setSecurityToken? Grid Service is a stateful service. This means that after the client sets a SecurityToken, any subsequent call will be applied to that SecurityToken.

Secure connections are not maintained on the server indefinitely, but are based on load conditions. The server will allow 30 unique secure connections to be set up for clients without any time limitations. As additional requests for secure connections are received by the server, connections will be released by the server on an ?oldest first? basis. No connection, however, may be released prior to 5 minutes after its creation.

If no SecurityTokens are passed in by the client, a non-secure Distributed LexEVS connection will be used. The server maintains one (and only one) un-secured Distributed LexEVS connection that is shared by any client not requesting security.

NOTE:

All non-secured information accessed by the LexEVS Grid Service is publicly available from NCICB and users are expected to follow the licensing requirements currently in place for accessing and using NCI EVS information.

LexEVS Data Grid Service API

The LexEVS Data Grid Service

The LexEVS Data Grid Service is a standard caGrid Data service based on the LexEVS 2009 Model

caGrid Data Service Documentation

For complete documentation on caGrid Data Services, see caGrid Data Service Documentation

Querying The System

To query the LexEVS Data Grid Service, use the standard caGrid CQL query method to compose queries. See caGrid Data Service API Documentation for more information.

Example LexEVS queries follow.

Query for a Concept with a specific Code

- Example: Concept: C12345

```
<CQLQuery xmlns="http://CQL.caBIG/1/gov.nih.nci.cagrid.CQLQuery">
    <Target name="org.LexGrid.concepts.Concept">
        <Attribute name="_entityCode" value="C12345" predicate="EQUAL_TO"/>
    </Target>
</CQLQuery>
```

Query for a Concept with a specific Presentation Text

- Example: A concept with a namespace 'SNOMED Clinical Terms' that contains a Presentation equal to 'Heart'

```
<ns1:CQLQuery xmlns:ns1="http://CQL.caBIG/1/gov.nih.nci.cagrid.CQLQuery">
    <ns1:Target name="org.LexGrid.concepts.Entity">
        <ns1:Group logicRelation="AND">
            <ns1:Association name="org.LexGrid.concepts.Presentation" roleName=
                <ns1:Group logicRelation="AND">
                    <ns1:Association name="org.LexGrid.commonTypes.Text" roleNa
                        <ns1:Group logicRelation="AND">
                            <ns1:Attribute name="_content" predicate="EQUAL_TO"
```

```
                        </ns1:Group>
                    </ns1:Association>
                </ns1:Group>
            </ns1:Association>
            <ns1:Attribute name="_entityCodeNamespace" predicate="EQUAL_TO" val
        </ns1:Group>
    </ns1:Target>
</ns1:CQLQuery>
```

Restrict Results to Specific Attributes

- Example: Retrieve all of the 'localIds' of any 'SupportedAssociation' in the system.

```
<ns1:CQLQuery xmlns:ns1="http://CQL.caBIG/1/gov.nih.nci.cagrid.CQLQuery">
    <ns1:Target name="org.LexGrid.naming.SupportedAssociation"/>
    <ns1:QueryModifier countOnly="false">
        <ns1:DistinctAttribute>_localId</ns1:DistinctAttribute>
    </ns1:QueryModifier>
</ns1:CQLQuery>
```